Interfacing to ADAT® Lightpipe Devices

On March 22nd 1994, Alesis patented “A method and apparatus for providing a digital audio interface protocol” (US Pat 5,297,181). This patent describes in some detail the protocol commonly known as ADAT® Lightpipe, capable of transporting 8 channels of 24-bit, 44.1/48 kHz audio up to 10m over a single fiber optic cable.

In this post, I describe a simple, compact ADAT receiver suitable for implementation in programmable logic devices.

Despite the popularity of this protocol in the music industry, I found it very difficult to obtain enough technical information to develop a working ADAT interface. Most of the patent is devoted to VCO clock extraction techniques, and two contradictory descriptions of the frame structure are provided (is the sync pattern at the end of the frame, or at the start?).

Luckily, it turns out the protocol itself is simply a combination of a few well-documented encoding and modulation schemes. The signal is NRZI-encoded (1=transition, 0=no transition), which helps with clock extraction. Data is transmitted using very simple 4B/5B code, meaning that a ‘1’ is inserted after every nibble of audio or status information. At some point in the frame, a sync pattern consisting of 10 ‘0’s is transmitted: since this is a longer period without transitions (NRZI-encoded ‘1’s) than is allowed by the 4B/5B encoding scheme, it is easy to recognise.

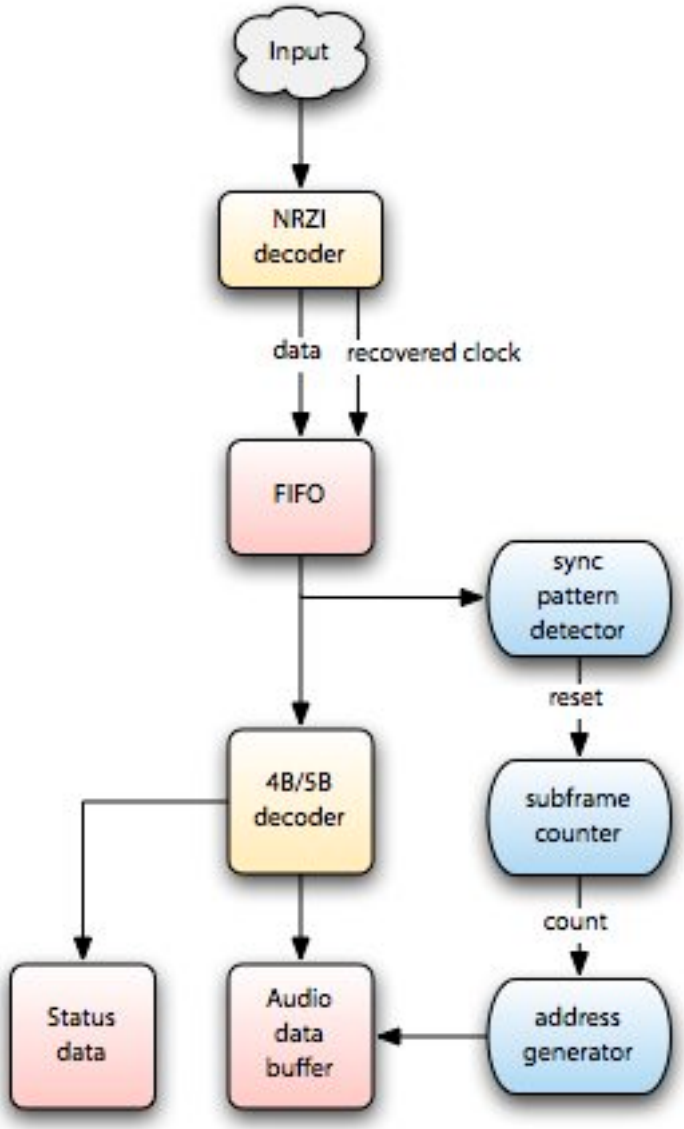
According to (one interpretation of) the patent, the frame will look like this:

```
10000000000

1ssss

1aaaa1aaaa1aaaa1aaaa1aaaa1aaaa
1aaaa1aaaa1aaaa1aaaa1aaaa1aaaa
1aaaa1aaaa1aaaa1aaaa1aaaa1aaaa
1aaaa1aaaa1aaaa1aaaa1aaaa1aaaa

1aaaa1aaaa1aaaa1aaaa1aaaa1aaaa
1aaaa1aaaa1aaaa1aaaa1aaaa1aaaa
1aaaa1aaaa1aaaa1aaaa1aaaa1aaaa
```

```
1aaaa1aaaa1aaaa1aaaa1aaaa1aaaa
```

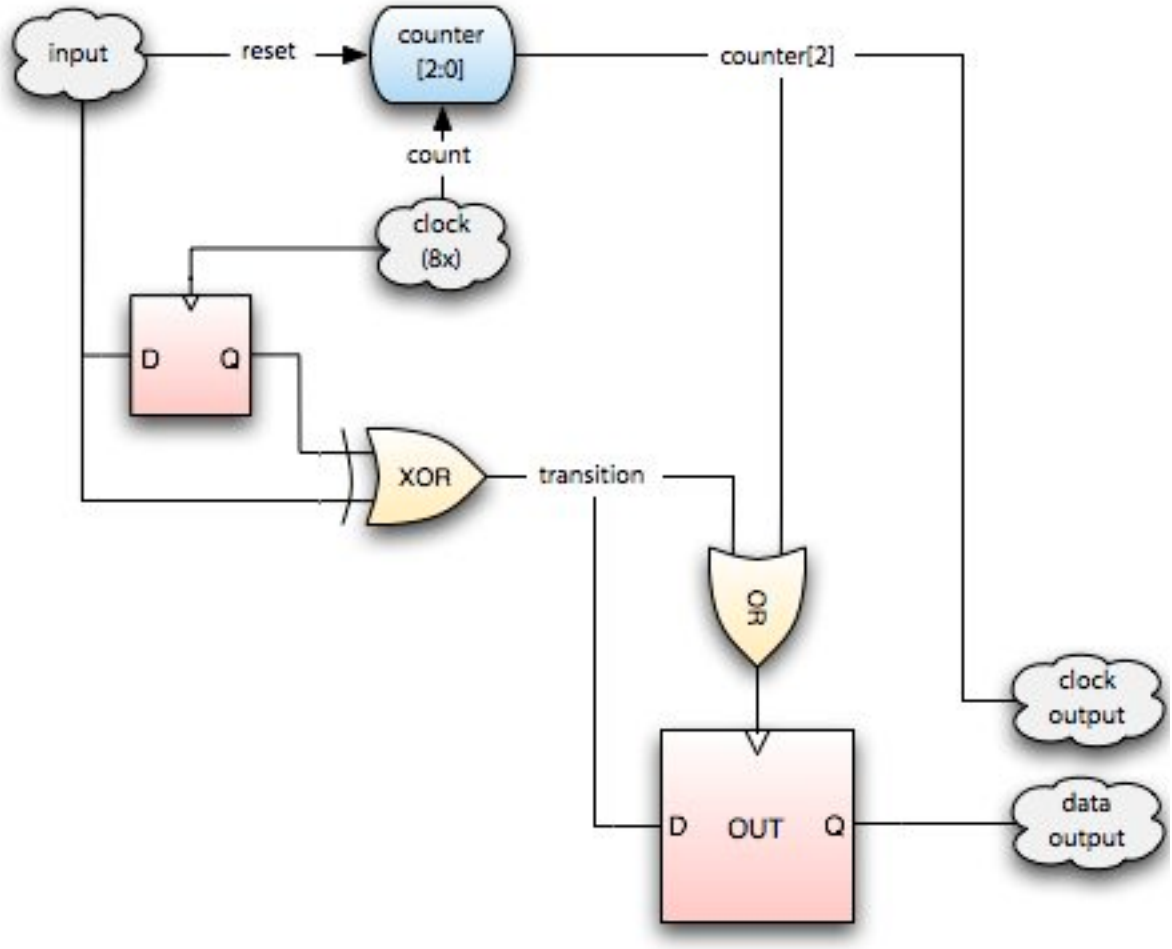
Where ‘a’ denotes an audio bit, ’s’ a status bit, and ‘0′ and ‘1′ their respective binary values.

Based on this understanding, I set out to design an ADAT-compatible receiver. One possible receiver block diagram is shown here:



Having been thoroughly confused by the patent’s description of VCO locking stages, charge pumps, and phase detectors, I realised that since there’s a guaranteed transition every 5 bits, the receiver can actually get away with less phase lock accuracy than a UART! A simple 8x oversampling design would be adequate, and the oversampling clock frequency could be a few percent off without

ill effect. A block diagram of the NRZI decoder/clock recovery is shown here:



According to the receiver block diagram above, the decoded data and recovered clock is passed to a FIFO: this is important, because the recovered clock will have significant jitter (particularly if the oversampling clock deviates much from the received data rate), and if the receiver is to form part of a larger system, it will need a much more stable clock source. Ideally the recovered clock signal would be fed to a PLL for cleaning up before being passed back into the receiver to clock the FIFO output. Obviously, it's not possible to clock the FIFO output with a clock derived from the oversampling clock, because there clock of the transmitting ADAT device and the local clock will differ in frequency. Even very small differences will cause buffer over-/under-runs, and result in data errors.

As it turns out, the rest of the system was even easier to design. The following modules were used:

- The 4B/5B decoder in the diagram is just a 10-bit shift register with parallel outputs

- The sync pattern detector looks for the pattern ‘10000000’ (since even five ‘0’s in a row is invalid anywhere other than in the sync pattern, and splitting the 16-bit sync pattern + status nibble section of the frame into two 8-bit sections makes timing much easier)
- The subframe counter is reset to 0 every time the sync pattern is found, and counts two subframes of 8 bits (sync, sync + status) then 24 frames of 10 bits (8x audio data)
- The address generator maps each subframe to a location in the output buffer, and generates a write enable signal at the end of each data subframe
- The output buffer takes the 8 data bits from the 4B/5B decoder (ignoring the two padding ‘1’s), and stores it in the appropriate location

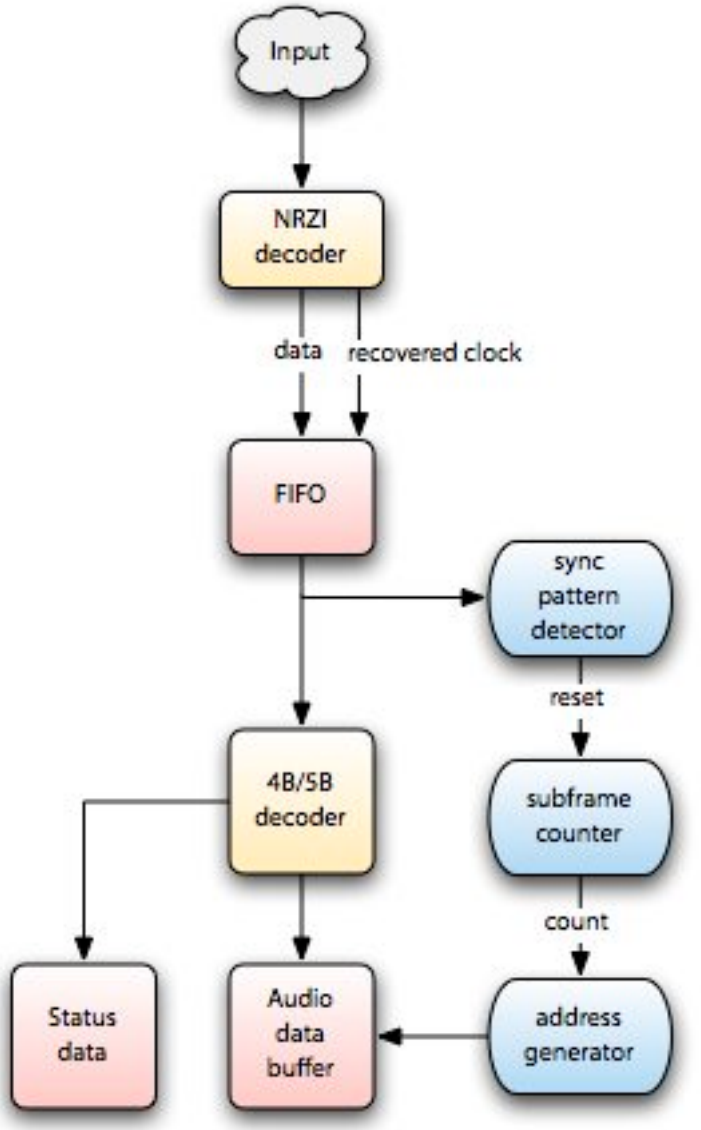
In order to test my designs, I considered a few options. Since I’m a software person at heart, I eventually decided to buy a basic FPGA development board (an Altera EP1C3-based Pluto-II from KNJN, USD49.00), and learn Verilog (mainly because the examples I saw looked prettier than equivalent VHDL code).

Interfacing to ADAT® Lightpipe Devices, Part II

In Part I of this series, I described in general terms a receiver for Alesis' ADAT Lightpipe protocol. This part will describe the implementation and testing of this receiver design on an Altera Cyclone EP1C3 FPGA development board (Pluto-II from KNJN).

Having learned a bit of Verilog, and successfully run through a few tutorials (Ooh, a flashing LED!), I turned to implementation of the designs outlined in my previous post. Thankfully, the simplicity of the project more than compensated for my complete lack of knowledge and education, and the code was completed within a few dozen hours.

The basic structure of the code follows this diagram:



To simplify the implementation, I opted to ignore the status bits (one is used for S/MUX, one is for a 32-bit timecode, and the other two are undefined), and limit the sample rate (Fs) to 48kHz only. My development board has a 25MHz crystal, so it's not possible to get a neat 8x 12.288MHz (256x Fs) for the oversampling clock; the actual frequency turns out to be 96MHz (2.5% off the 98.304MHz ideal). As it turns out, that doesn't matter much. What matters more is that I didn't have an external PLL handy (or a spare PLL inside the FPGA), so I had to derive the system clock from the ADAT data stream. That's a really bad idea, but more on that in a later post.

The code was written in ordinary Verilog, except for the FIFO and output buffer modules, which were generated by wizards in the Quartus II software. That made life easier, because there are all sorts of timing and synchronisation issues involved in the design of FIFO buffers and dual-port RAM, and the Cyclone device has built-

in support for both of them.

I ended up splitting the implementation into five sub-modules:

- nrzi_decoder: takes an ADAT signal from an FPGA input pin, and outputs recovered clock and recovered data;
- fifo: input data and clock provided by nrzi_decoder, output clock derived from the system clock (ouch!);
- frame_detector: combination shift register (4B/5B decoder) and sync pattern detector, outputs 8 bits of decoded audio data at a time, along with subframe ready/frame start signals;
- adat_stream_in: ties nrzi_decoder, fifo, and frame_detector together for a single ADAT input;
- adat_i_frame_buf: combines outputs of up to eight adat_stream_in modules, and includes subframe counter and address generator logic, plus the audio data buffer itself. The address generator output is just a combinatorial function of the subframe count and current input stream.

Based on simulation of a manually-constructed ADAT Lightpipe waveform in Quartus II, it appeared the receiver logic “worked” (after a bit of debugging); total resource usage for four inputs (that’s 32 channels) was 4 FIFOs and around 500 LEs, plus the output buffers (more on that in a later post). Of course, creating and manually verifying the simulation input was tedious, I only had three frames of data to test on (luckily the single-Fs receiver doesn’t need time to lock), and all it actually proved was that my code implemented my own interpretation of the ADAT protocol. The real test of the design could only come through implementation in hardware.

Interfacing to ADAT® Lightpipe Devices, Part III

Parts I and II of this series relate the design and Verilog implementation of a multichannel ADAT Lightpipe receiver.

Obviously there's not much point writing about a totally untested design, so in this post I'll show you the actual hardware (such as it is…), my test setup, and conclude the receiver module development.

As previously stated, the design is based around the Pluto-II from KNJN. The main reason for choosing this board was price: at USD49.00 plus shipping, I figured it'd make a something-or-other even if this project didn't work out. Of course, if I was buying again, I'd go for something like the Altera Cyclone 2C20 Starter Kit—three times the price, but with USB programming support (really useful, RS-232 is getting hard to find these days), six times the FPGA capacity, embedded 18×18 multipliers in the FPGA, more IOs, more ports, more everything.

In order to get my board interfaced to an ADAT Lightpipe device, I had to buy a few TOSLINK connectors. Most places seem to suggest the TORX143 part from Toshiba, but I couldn't find a source—I ended up getting the TORX177, which seems to be better in every respect. The versions with built in shutters are cool, but a word of warning: if you break the shutter (oops!), the TOSLINK cables will no longer clip in.

My parts were purchased from Digi-Key; shipping to Australia was about USD30.00, so I ordered ten each of the TORX177 (receiver) and TOTX177 (transmitter) parts to make it worthwhile. Australians might have some luck at Jaycar, although I'm not sure what the specs on their TOSLINK parts are. The data rate for ADAT is 12.288MHz, which is significantly above that of S/PDIF (the intended application of most of these devices).

Assembly of the device was simple: I soldered 0.1″ header to the FPGA board, soldered sockets to a prototype board, and worked off the Toshiba reference design for the TORX177 (47uH inductor on Vcc, 0.1uF decoupling cap). The TORX177 is a 5V part, and the EP1C3 is a 3.3V part with non-5V-tolerant inputs, so a voltage divider to convert levels would be wise.

A couple of pictures of my protoboard assembly (including input, output, and MIDI

IN):


(Since assembling this board, I've designed a couple of custom PCBs carrying 4x TORX177 and 4x TOTX177 along with decoupling capacitors and the inductors. I'll post some pictures of those once the prototypes arrive from BatchPCB.)

My test setup consisted of the prototype, a Behringer (yeah, I know…) ADA8000 AD/DA converter with ADAT Lightpipe I/O based on the Wavefront Semiconductor ADAT chipset, a Tascam FW-1804 analog+ADAT FireWire interface, and my PowerBook.

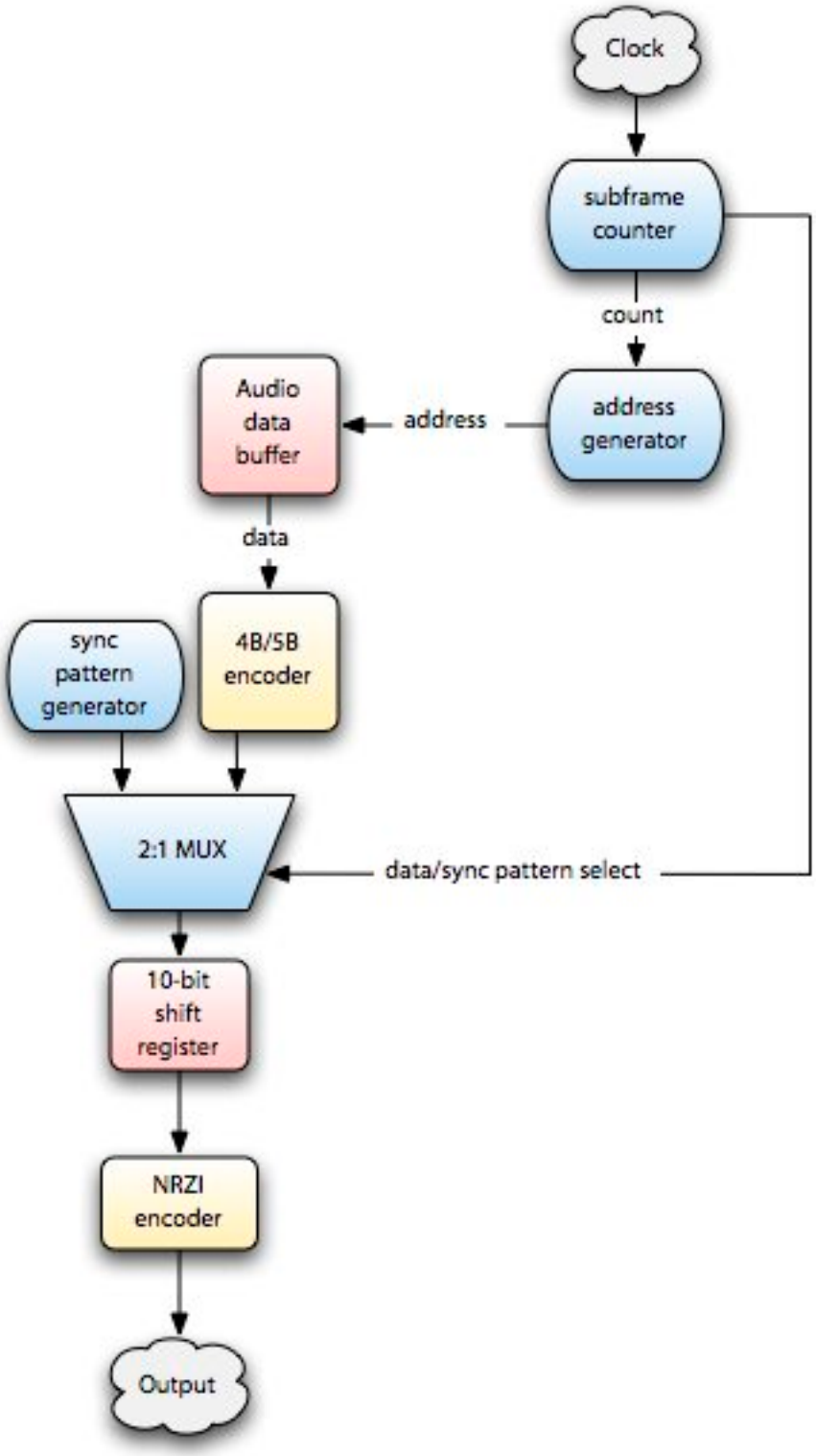
The first thing I discovered when I started testing was that my interpretation of the protocol was basically correct. It ultimately didn't matter whether sync patterns were expected at the start or the end of the frame, and the 4B/5B coding was enough to ensure a consistent lock even when the sampling clock differed from the data rate by 2.5% (96MHz actual vs. 98.304MHz ideal).

The next thing I discovered was a series of logic mistakes and misunderstandings of basic Verilog constructs. Oops.

With those issues out of the way, the device synced correctly to ADAT signals at 48kHz, and read the right data on the right channels, proving the ADAT protocol is within easy reach of homebrew FPGA-based audio projects.

Interfacing to ADAT® Lightpipe Devices, Part IV

To complement the ADAT Lightpipe receiver core (see Parts I, II and III), I developed an ADAT Lightpipe transmitter core, organised along similar lines:



This core is similar to the ADAT receiver, but works in reverse:

- A clock signal (in my implementation, this is derived from the system clock) is passed to the subframe counter, which generates a subframe index and bit index output. For simplicity, I divide the ADAT frame into two 8-bit sync

subframes, plus 24 10-bit data subframes.

- The subframe index from the counter is passed to an address decoder, which generates the audio buffer look-up addresses for the current frame.
- The 4B/5B encoded buffer output and the preset sync word (see Part I) form the input to a 2:1 multiplexer, which selects either sync or data depending on the subframe index from the counter.
- The output of the mux is loaded into a shift register at the end of every subframe (based on the bit index output of the subframe counter).
- The output of the shift register is NRZI encoded, and passed out to the external TOSLINK output device (in my case, a TOTX177)

While I designed the ADAT receiver to scale from 1-8 ADAT inputs because of the RAM block and LUT requirements of each input stream, the ADAT transmitter requires almost no extra resources (<12 LUTs) per stream beyond the first. Thus, the core modules are all designed to cope with 8 streams:

- adat_o_frame_gen: shares subframe counting/address decoding logic with the ADAT receiver; handles the audio buffer and data/sync mux; also generates load/shift enable signals for the 10-bit shift registers
- sreg_10 (x8): 10-bit parallel-load shift registers, each of which handles a single ADAT output stream
- nrzi_encoder (x8): NRZI encodes each ADAT output stream

As implemented, the ADAT transmitter and receiver cores use a double-buffer arrangement that enables the two RAM blocks involved to be used simultaneously as the output buffer of the receiver and the input buffer of the transmitter (perhaps with some address modification between, to provide routing functionality). The generated output in this configuration is very sensitive to jitter in the input, but if the modules use an internal crystal oscillator, and other ADAT devices are set to sync to the output signal, good results can be achieved.

A 64×64 channel ADAT® Lightpipe Router/Mixer

To test my ADAT Lightpipe receiver and transmitter modules (see Parts I, II, III and IV), I decided to build a 64×64 channel router/mixer. Since the death—one week out of warranty—of my Behringer DDX3216 (great concept, great price, crappy execution), I've been looking for a similarly flexible digital mixer, and while the Yamaha gear (O1V96, LS9) is nice, their entry-level gear is limited in the number of digital channels you can work with.

As the photos in Part III show, the prototype used MIDI for routing and mix level changes. The advantages of MIDI are the large number of devices that use it, the simplicity of the electrical interface, and the ridiculous ease of interpreting MIDI messages. The main disadvantage is the slow baud rate and the limited number of available channels and CCs (doing everything via SysEx would remove most of the advantages).

The key things I needed to add to the ADAT transmitter/receiver design were some means of passing the audio from one core to another, and mixer/router modules. In my initial idealistic zeal, I over-engineered the system; because I had originally intended to write a number of audio processing modules (mixer, biquad-based parametric EQ, dynamics control), I used a design loosely based on that of ProTools, with a 256-channel 32-bit TDM bus, and input, output and processing cores hanging off it. While that sort of arrangement is good for flexibility, the resource use (RAM blocks in particular) is suboptimal. If I were to re-write the system, I'd probably go with a system in which the (double-buffered) output of one module serves as the input buffer of the following module.

Either way, after the prototype was made, tested, and broken (those silly shuttered TOSLINK connectors…), I looked around online for a more capable platform. Because of the limited block RAM of the EP1C3, and the lack of embedded multipliers (it's the oldest, smallest member of the Altera Cyclone family), it was clear that the KNJN boards would only support 8×8 configurations.

After looking at a number of solutions—including various Xilinx boards, and the Altera development boards—I decided to go with the TS-7300, an embedded ARM9/FPGA board from Technologic Systems. Although a larger FPGA could have supported a soft core running Linux, the TS-7300 seemed to be a more flexible solution in that the on-board EP2C8 FPGA can be re-configured on the fly (in all of

0.1–0.5 sec). This makes it easy to reset the device to a known state, useful for debugging.

In the meantime, my custom TOSLINK rx/tx PCBs had arrived, so I assembled them. It was my first time doing SMD soldering, but it's hard to go wrong with the 1812 and 1210 size parts (inductors and caps). Fortunately, I realised before I powered up that I'd switched the TOSLINK interface pins in the PCB layout, so I disassembled them all and soldered them to the underside of the PCB. Must watch that one in future revisions…

* ADAT is a registered trademark of Alesis Corporation.